IN THE SPECIFICATION

Please replace the paragraph beginning on Page 3, line 8 through Page 2, line 2, with the following amended paragraph.

The storage device 204 includes a ROM (Read-Only Memory), a RAM (Random-Access Memory) and an EEPROM (Electric Erasable Programmable Read-Only Memory). The ROM is a memory not allowing information stored therein to be altered. The ROM is used mainly for storing a program. On the other hand, the RAM is a memory (such as data memory 206) allowing data stored therein to be rewritten with a high degree of freedom. If power supplied by a power supply to the RAM is turned off, however, data stored in the RAM is lost. Thus, when the IC card 101 is removed from the reader and writer, data stored in the RAM is lost since power supplied by the power supply of the reader and writer to the RAM is cut off. The EEPROM is a memory for storing information that needs to be updated but must be retained even if the IC card 101 is pulled out from the reader and writer. In the case of a prepaid IC card 101, for example, information stored in the EEPROM includes the number of times the IC card 101 have been used so far. Such information needs to be updated each time the IC card 101 is

used and needs to be retained in the EEPROM even if the IC card 101 is pulled out from the reader and writer.

Please replace the paragraph beginning on Page 22, line 5 through line 18, with the following amended paragraph.

For such processing, data for disturbance is generated in such a way that the hamming weight of the data for disturbance is equal to half the bit count of the data for disturbance, and the appearance probability of the logic value 0 or 1 at each bit position of the data for disturbance is set at 0.5. As a result, it is no longer easy to identify the data for disturbance from the waveform of a current consumed during processing of the data for disturbance. It should be noted that the probability of appearance does not need to be set at strictly 0.5. That is to say, the probability may be smaller or greater than 0.5. However, an appearance probability of 0.5 is desirable. The closer the probability of appearance to 0.5, the more desirable the probability.

Please replace the paragraph beginning on Page 26, line 21 through Page 27, line 16, with the following amended paragraph.

There are several techniques for generating random numbers having uniform and constant hamming weights. Fig. 6 is a diagram showing a data flow of a first embodiment implementing a technique to generate random numbers having constant uniform hamming weights. In this embodiment, the number of bits in a random number to be generated is 2n. As shown in the figure, first of all, an n-bit-random-number generator 601 generates an n-bit random number 602. The n-bit-random-number generator 601 may generate a pseudo random number or a true random number which is selected from results of measurement of a physical phenomenon. Then, a bit-inverting operation method 603 is used for inverting the generated n-bit ~~ransom~~ random number 602 to produce an inverted n-bit ~~ransom~~ random number 604. Subsequently, a data concatenation method 605 is used for concatenating the n-bit random number 602 and the inverted n-bit random number 604 to generate a constant-hamming-weight 2n-bit random number 606. This is because, if the number of bits each having the logic value 1 in the n-bit random number 602 is n1 and the number of

bits each having the logic value 0 in the n-bit random number 602 is n2, then the following equation holds true:

$$n1 + n2 = n \quad \text{(Eq. 17)}$$

Please replace the paragraph beginning on Page 28, line 3 through line 24, with the following amended paragraph.

Fig. 7 is a flowchart representing a second embodiment implementing a technique to generate random numbers having constant uniform hamming weights. As shown in the figure, the random-number generation represented by the flowchart begins, after a start at 701, with a step 702 at which a target hamming weight H is input. Then, at the next step 703, a random number R is generated. Subsequently, at the next step 704, the hamming weight RH of the generated random number R is computed. The flow of the random-number generation then goes on to a step 705 to form a judgment as to whether or not the hamming weight RH of the generated random number R is equal to the target hamming weight H. If the hamming weight RH of the generated random number R is not equal to the target hamming weight H, the flow of the random-number generation goes back to the step 703 at which another random number R is generated. If the hamming weight RH of the generated random number R is

equal to the target hamming weight H, on the other hand, the flow of the random-number generation goes on to a step 706 at which the random number R is passed to a calling routine as a return value. Then, at the next step 707, the generation of random numbers is ended.

Please replace the paragraph beginning on Page 28, line 25 through Page 30, line 9, with the following amended paragraph.

Fig. 10 is a flowchart representing a third embodiment implementing a technique to generate random numbers having constant uniform hamming weights. First of all, pieces of m-bit data having uniform constant hamming weights are collected in a table. The embodiment generates only random numbers that have uniform constant hamming weights and each have a bit count equal to a multiple of m. As shown in the figure, after a start at 1001, the random-number generation represented by the flowchart begins with a step 1002 at which the bit count of a random number to be generated is set at n. Then, at the next step 1003, a result of division of n by m is substituted for L. In the basic flow of the random-number generation, L m-bit random numbers having uniform constant hamming weights

are generated and concatenated to generate an n-bit random number having a constant hamming weight. Subsequently, at the next step 1004, a variable D for accommodating the n-bit random number being generated to have a constant hamming weight is initialized at 0. Then, at the next step 1005, a random number R is generated. Subsequently, at the next step 1006, a piece of m-bit data having a constant hamming weight is fetched from the table cited above. The piece of m-bit data fetched from the table is indicated by an index having a value equal to the random number R. The piece of m-bit data fetched from the table is stored in a variable d. Subsequently, at the next step 1007, the variable D is shifted to the left by m bits and then the variable d is added to the variable D. The pieces of processing from the step 1005 to generate a random number R to the step 1007 to add the variable d to the variable D shifted to the left by m bits are carried out repeatedly L times. A step 1008 is adopted to form a judgment as to whether or not the pieces of processing have been carried out repeatedly L times. If the pieces of processing have been carried out repeatedly L times, the flow of the random-number generation goes on to a step 1009 at which the variable D is passed to a calling routine as a return value, and then to end at 1010.

Please replace the paragraph beginning on Page 31, line 7 through Page 32, line 25, with the following amended paragraph.

As shown in Fig. 8, the creation of a list begins, after a start at 801, with a step 802 at which the hamming weight is stored in a variable Hamming. Then, at the next step 803, the bit count is stored in a variable MaxBit. Subsequently, at the next step 804, an array pos [j] where j = 0 to (Hamming - 1) is initialized at values indicating the positions of bits in a bit array which each have a logic value of 1. A bit position can be any value in the range 0 to (MaxBit -1). Then, at the next step 805, an index num pointing to a slot in the bit-array list dat is initialized at 0. The dat bit-array list's slot pointed to by the index num will be used for storing a computed bit array at the next step 806. In addition, an index b used as the subscript of the array pos [b] in the following processing is initialized at -1. Subsequently, at the step 806, the bit array is computed and stored in the dat bit-array list' slot pointed to by the index num. Then, at the next step 807, the index num is incremented by 1. Subsequently, at the next step 808, the index b used as the subscript of the array pos [b] is incremented by 1. The

flow of the list creation then goes on to a step 809 to form a judgment as to whether or not the script b has not reached (Hamming -1), which is a subscript value corresponding to the bit array's highest-order bit having the logic value of 1. That is to say, the judgment is formed to determine whether or not pos [b] does not have the value indicating the position of the highest-order bit having the logic value of 1 in the bit array. If the subscript b has reached (Hamming -1), the flow of the list creation goes on to a step 812. If the subscript b has not reached (Hamming -1), on the other hand, the flow of the list creation goes on to a step 810. At the next step 810, the bit array's higher-order bit position, that is, (pos [b] + 1), is checked to form a judgment as to whether or not the bit at the higher-order position or the bit at (pos [b] + 1) already has the logic value of 1. If the bit at the bit array's higher-order position already has the logic value of 1, the flow of the list creation goes on to the step 812. If the bit at the bit array's higher-order position or the bit at (pos [b] + 1) has the logic value of 0, on the other hand, the flow of the list creation goes on to a step 811. At the step 811, the logic value of 1 in the bit array is shifted from the bit position p [b] to the bit position (p [b] +1) and the flow

of the list creation then goes back to the step 806 to create another bit array.

Please replace the paragraph beginning on Page 36, line 1 through line 15, with the following amended paragraph.

The embodiment shown in Fig. 9 adopts the same technique to create a list of bit arrays having constant uniform hamming weights as the embodiment shown in Fig. 8. The embodiment shown in Fig. 9 is different from that of Fig. 8 in that, in the case of the embodiment shown in Fig. 9, after steps 901-906 (which are similar to steps 801-806), the disturbance data created on the list is not used as it is but processed by using a disturbance-data-processing method. The hamming weight of the processing results is stored in a valuable hxdat at a step 907 of the flowchart shown in Fig. 9. Then, the flow of the list creation goes on to a step 908 to form a judgment as to whether the hamming weight is unchanged only if the hamming weight is found unchanged is the data for disturbance is cataloged on the bit-array list dat. The remaining steps 909-921 of the flowchart is the same as steps 807-819 of the flowchart shown in Fig. 8.